Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Caustics Rendering and Synthetic Data

Bachelor Thesis

2024
Michal Kubirita

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Caustics Rendering and Synthetic Data

Bachelor Thesis

Study Programme:	Applied Computer Science
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	Mgr. Lukáš Gajdošech
Consultant:	doc. RNDr. Martin Madaras, PhD.

Bratislava, 2024
Michal Kubirita

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Michal Kubirita
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Synthetic Dataset Rendering with Parametric Transparent and Translucent Objects
Generovanie syntetických dát s parametrickými priesvitnými a priehľadnými objektami

Anotácia: K tréningu robustnej neurónovej siete potrebujeme prístup k trénovacím dátam. Konvolučná neurónová sieť vie byť natrénovaná metódami hlkbokého učenia s použitím syntetických, alebo reálnych anotovaných dát, za účelom spracovania mračien bodov ako napr. Filtrovanie alebo odhad pózy objektu. Pokiaľ nie sú k dispozícii reálne a anotované dáta, syntetické datasety môžu byť renderované a použité na tréning. Hlavným cieľom práce bude navrhnutie renderovacieho systému na renderovanie scén zložených z priesvitných a priehľadných predmetov. Vygenerované dáta budú neskôr použité na tréning neurónovej siete na spracovanie mračien bodov. Priehľadné a priesvitné objekty by mali byť parametrizované za účelom randomizácie v generovanej scéne. V optimálnom prípade, renderovanie chceme mať v reálnom čase, preto chceme použiť Unreal Engine. Alternatívne sa môže použiť Blender s LuxCore rendererom.

Cieľ:
- Presúmať možnosti renderovacích a herných enginov ako Unreal Engine a LuxCoreRender v Blendery za účelom renderovania priehľadných a priesvitných objektov
- Prioritizujte rýchlejšie renderovanie, ideálne v reálnom čase, preto sa zamerajte primárne na Unreal Engine
- Vytvorte parametrický model transparentných a translucentných objektov, ktoré sa môžu parmetrizovať cez skript a vkladať do scény
- Renderujte scénu s kaustikami, s parametrickými modelmi, prípadne s existujúcimi priesvitnými a priehľadými modelmi z UE a porovnajte výsledky z UE s výsledkami z Blenderu

Literatúra:
Rui Wang, Wei Hua, Yuchi Huo, Hujun Bao 2022, Real-time Rendering and Editing of Scattering Effects for Translucent Objects, https://doi.org/10.48550/arXiv.2203.12339
Unreal Engine documentation, https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Materials/HowTo/Transparency/
Unreal Engine forums, https://forums.unrealengine.com/t/transparent-materials-render-infront-of-closer-translucent-materials/278592
Unreal Engine RTX branch, https://developer.nvidia.com/game-engines/unreal-engine/rtx-branch

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

https://forums.unrealengine.com/t/inside-unreal-dlss-and-rtxgi-with-nvidia/149916

Kľúčové slová: synthetické dáta, priehľadné a priesvitné objekty, virtuálna scéna, Unreal Engine

Vedúci: Mgr. Lukáš Gajdošech
Konzultant: doc. RNDr. Martin Madaras, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 04.10.2023

Dátum schválenia: 04.10.2023 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Michal Kubirita
Study programme: Applied Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study: Computer Science
Type of Thesis: Bachelor´s thesis
Language of Thesis: English
Secondary language: Slovak

Title: Synthetic Dataset Rendering with Parametric Transparent and Translucent Objects

Annotation: Training of robust neural networks is based on access to the training data. A convolutional neural network can be trained in a deep learning manner using synthetic or real captured annotated data to perform point cloud processing tasks, such as filtering the point clouds or pose estimation of an object. If there is no available real annotated training dataset a synthetic one can be rendered and used. The main scope of the thesis would be to propose a rendering pipeline for synthetic scans composed of transparent and translucent objects. The generated data will be later used to train a neural network for point cloud processing. The translucent and transparent objects should be parametrized in order to be randomized in the scene. In an optimal scenario, the rendering of the scene should be performed in real-time, therefore Unreal Engine should be used. Alternatively, a Blender with LuxCoreRenderer can be used.

Aim: - Explore the possibilities of rendering and game engines such as Unreal Engine or Blender's LuxCoreRender for rendering translucent and transparent objects
- Prioritize faster, real-time rendering, therefore focus mainly on Unreal Engine
- Create a parametric model of translucent and transparent objects that can be parametrized and added to a scene using a script
- Render the scene with all the caustics, with parametric models and assets from UE and compare the results rendered in UE with results from Blender

Literature: Rui Wang, Wei Hua, Yuchi Huo, Hujun Bao 2022, Real-time Rendering and Editing of Scattering Effects for Translucent Objects, https://doi.org/10.48550/arXiv.2203.12339
Unreal Engine documentation, https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Materials/HowTo/Transparency/
Unreal Engine forums, https://forums.unrealengine.com/t/transparent-materials-render-infront-of-closer-translucent-materials/278592
Unreal Engine RTX branch, https://developer.nvidia.com/game-engines/unreal-engine/rtx-branch
https://forums.unrealengine.com/t/inside-unreal-dlss-and-rtxgi-with-nvidia/149916

Keywords: synthetic dataset, translucent and transparent objects, virtual scene, Unreal Engine

Supervisor: Mgr. Lukáš Gajdošech

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

Consultant:	doc. RNDr. Martin Madaras, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	04.10.2023
Approved:	04.10.2023

doc. RNDr. Damas Gruska, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: I am sincerely grateful for the unwavering moral support from my friends.

Abstrakt

Slovenský abstrakt v rozsahu 100-500 slov, jeden odstavec. Abstrakt stručne sumarizuje výsledky práce. Mal by byť pochopiteľný pre bežného informatika. Nemal by teda využívať skratky, termíny alebo označenie zavedené v práci, okrem tých, ktoré sú všeobecne známe.

Kľúčové slová: jedno, druhé, tretie (prípadne štvrté, piate)

Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

Contents

List of Figures

List of Tables

Chapter 1

Background

In computer vision and computer graphics, translucent objects are a problematic subject. Rendering them realistically needs nontrivial algorithms. In case of making 3D scans of these objects, it is hard to capture them and segmentation is difficult due to indeterminate boundaries. Results of these operations are often imperfect, which is caused by reflection and refraction of light shined on transparent surfaces.

To render these objects realistically we need to consider physical interactions of light. Transparent materials behave differently compared to the opaque ones. Rays of light coming through them create illuminated effects called caustics. This happens also inside transparent fluid bodies (e.g. water) but we will only focus on solid objects like bottles and glasses, which is referred to as mesh caustics [Yang and Ouyang, 2021].

This chapter covers related issues and known solutions. Firstly, we will describe some rendering techniques and then elaborate on the subject of caustics.

1.1 Rendering Techniques

In 3D graphics, rendering is the process of transforming a virtual 3D scene into a digital image. Rendering engines (renderers) use various methods to visualize and illuminate a 3D scene. We outline some basic terms and methods below, gradually moving to more advanced ones.

Offline vs Online Rendering

Offline rendering is sometimes called pre-rendering, since its result is an already finished image, a render, that may have taken hours to create. Its main priority is photorealism with accurate physical lighting. It is used in films and static images demanding high image quality.

Online rendering is also called real-time or interactive rendering because of its high performance. Its main goal is performance - online renders take only a few milliseconds

to make giving programs the ability to be visually interactive. It is used mainly in games.

1.1.1 Rasterization

A common implementation of online rendering is rasterization, a process of rasterizing scene objects into pixels. This is done by splitting the geometry of objects into triangles. Those are sorted and culled with visibility algorithms. Triangles are also divided into fragments representing at most one pixel. The color of each pixel in the image is determined by fragment shaders which are applied to the fragments. [Hughes, 2014]

Rasterization is able to generate many frames per second. However, it is not suited for photo-realism. It requires a lot algorithmic augmentations, tricks and effects to create realistic images.

1.1.2 Ray Casting

According to Haines and Akenine-Möller [2019], "ray casting is the process of finding the closest, or sometimes just any, object along a ray." In computer graphics, ray casting has many applications.

In the context of rendering, a ray is cast from the camera (also called the eye) through a pixel in the image plane. This type of ray is called a viewing ray and other times just a view ray [Glassner, 1989; Shirley et al., 2009; Haines and Akenine-Möller, 2019]. It travels until it hits the first object occluding the ray's path. For the purpose of shading the hit point, another ray can be cast into a light source to find out whether the object is in a shadow. Figure 1.1 shows a diagram visualizing the ray casting algorithm.

1.1.3 Ray Tracing

Ray tracing is an implementation of offline rendering. Its history is well-documented in the book Ray Tracing Gems by [Haines and Akenine-Möller, 2019]. The algorithm uses ray casting recursively to determine the color of each pixel in the image plane. The recursion limit is set as a maximum number of light bounces. It is the number of hit points (intersections) one ray can possibly achieve.

The recursion occurs in the intersections. After hitting a surface, child rays may be cast to simulate reflection and refraction. The directions of child rays are determined by the surface's material. Each of these rays can also be tested for shadow occurrence. When a ray reaches its limit, the color of the pixel is estimated by accounting the viewing ray, shadow rays and reflection and refraction rays [Haines and Akenine-Möller, 2019]. For a visualization see Figure 1.2.

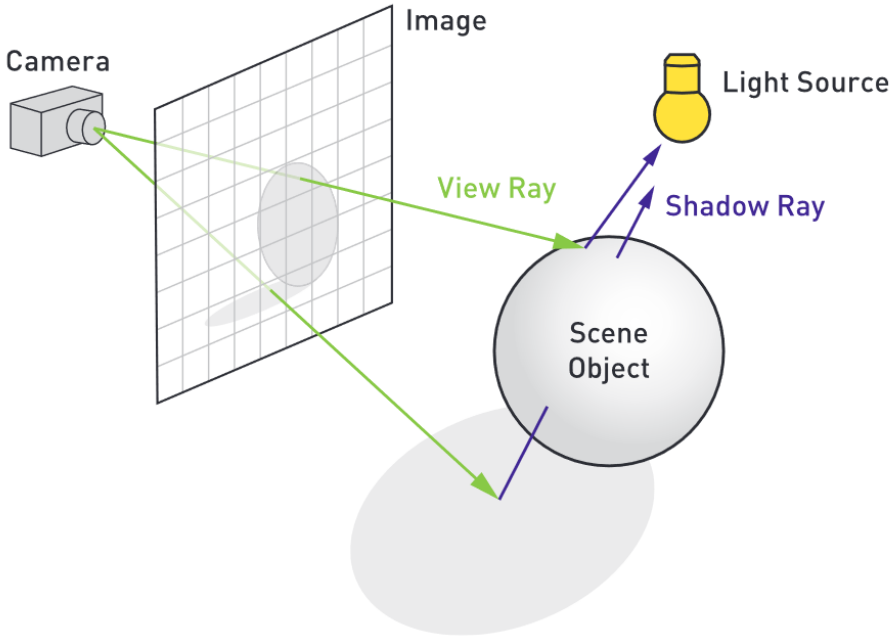

Figure 1.1: *Ray casting.* The diagram depicts hit point intersections of view rays. The hit point is shadowed if a shadow ray intersects an object on its way to a light source. *From Ray Tracing Gems by [Haines and Akenine-Möller, 2019].*

A more advanced implementation casts many rays per pixel to make the object edges smoother, create soft shadows or effects like camera depth of field [Shirley et al., 2009]. This improvement has been proposed by [Cook et al., 1984] and is called distributed ray tracing.

Spawning just one reflection ray from the hit point on a surface creates a mirrored reflection. To achieve a glossy look of shiny and polished materials, we must spawn multiple rays in a conical shape around the hit point and blend their results [Haines and Akenine-Möller, 2019].

For a comprehensive explanation of the fundamentals see the very first book about ray tracing [Glassner, 1989] and for more advanced and current topics see [Haines and Akenine-Möller, 2019; Marrs et al., 2021].

Forward and Backward Raytracing

We need to realize that the ray tracing process described above is reversed from the actual physical behaviour of light. In real world, light travels from a light source through the scene into the eye.

The light source emits photons in many directions. If we traced them *forward* the way they travel, we would find out only a few can be caught by our eye. Even then, some may have taken an overly complex route, arriving weaker from all the bounces. Thus, it is reasonably more efficient to shoot rays from the eye into the scene, tracing

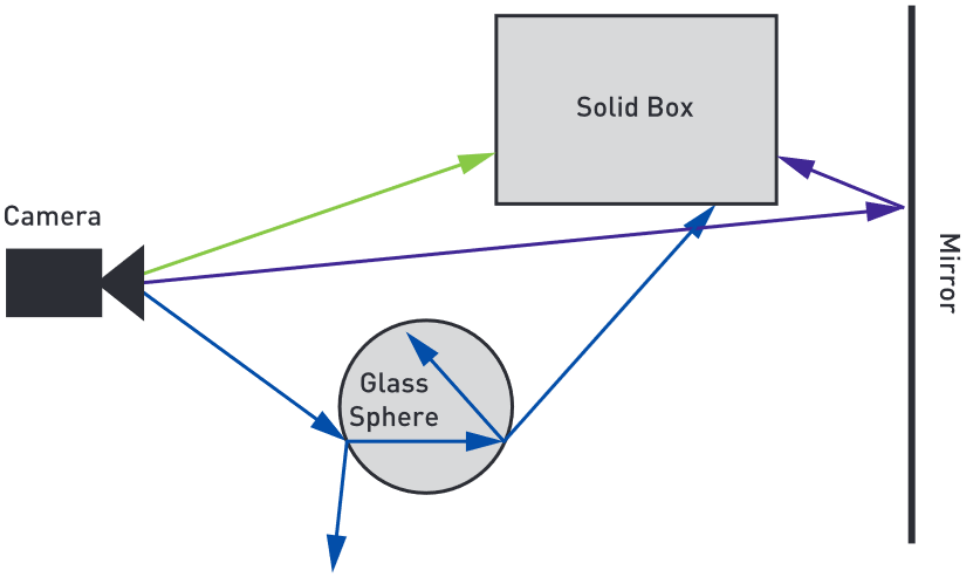

Figure 1.2: *Ray tracing.* The diagram depicts how a view ray forms child rays. The glass sphere as a translucent object reflects and refracts the rays, while the mirror only reflects them. As a result, the solid box affects reflections on the sphere and the mirror alike. *From Ray Tracing Gems by [Haines and Akenine-Möller, 2019].*

them *backwards*. These are guaranteed to hit the objects visible in our field of view (FOV). [Glassner, 1989]

1.1.4 Path Tracing

Path tracing is built upon the ray tracing algorithm. It uses Cook's sampling technique, each pixel is traced many times to light it more accurately. Furthermore, matte surface lighting is computed differently, matching real-world behaviour. To compute reflection of a matte or a diffuse surface, reflections rays scatter in all directions [Haines and Akenine-Möller, 2019].

When a ray hits this type of surface, a direction of a reflected child ray is chosen in a precise way. The child ray may reflect again, spawning a new ray recursively, until it finally reaches a light source. The more we repeat this process the more accurate the radiance of a pixel will be.

The sequence of ray intersections from the eye to a light source is called a path, thus the name path tracing. This style of rendering is credited to Kajiya [1986]. In the book Ray Tracing Gems Haines and Akenine-Möller [2019] say, "path tracing can, with proper care, give an unbiased result, one matching physical reality."

Bidirectional Path Tracing

Bidirectional path tracing extends the techniques above, accounting for the complex transport of light. This method was proposed by Lafortune and Willems [1993] and independently by Veach and Guibas [1995].

While traditional path tracing shoots rays solely from the camera's viewpoint, bidirectional path tracing traces paths from both the camera and light sources. In result, lighting effects like caustics can be achieved. [Hughes, 2014]

Connections are made between hit points of a random view sub-path and those of a random light sub-path. This gives us additional samplers and the ability to find caustic paths using connections to camera. This is not possible in regular path tracing. [Haines and Akenine-Möller, 2019]

1.1.5 The Rendering Equation

The Cook's and Kajiya's rendering methods require a way to efficiently choose rays for sampling. They use Monte Carlo Integration algorithm with probability density functions, stochastically distributing rays of light to solve this issue [Haines and Akenine-Möller, 2019]. This technique has been called distributed ray tracing [Glassner, 1989].

Below is a transport Equation 1.1 in an energy-balanced form, in computer graphics commonly referred to as the rendering equation [Haines and Akenine-Möller, 2019].

$$L_0(P, \omega_0) = \int_{S^2} f(P, \omega_0, \omega_i)\ L_i(P, \omega_i)\ |\cos\theta_i|\ d\omega_i \tag{1.1}$$

In the Equation 1.1, L_0 is the light leaving from surface point P in direction ω_0. Function f represents a bidirectional reflectance distribution function which describes reflection at point P. L_i is the incoming light in direction ω_i calculated recursively. θ_i is an angle between the surface normal and the incoming light direction. [Haines and Akenine-Möller, 2019] By integrating through all the light sources and surfaces, we get the resulting ray's radiance.

Now we can clearly see why is the ray's color only approximated. The integral could be hardly solved for each surface. And that is only for one view ray. Solving it using the Monte Carlo Integration made the advanced rendering techniques like path tracing possible to implement.

Monte Carlo Integration

Monte Carlo is a probabilistic method for approximating definite integrals. It involves generating random samples from a probability distribution over the domain of the integral and using these samples to estimate the integral value. [Shirley et al., 2009]

It is proven to be an effective solution to integration in rendering [Haines and Akenine-Möller, 2019]. Using Monte Carlo methods allows us to choose significant light rays to calculate radiance of a surface point. Equation 1.1 can then be solved as an estimation.

Equation 1.2 is an example showing how to solve an n-dimensional integral of function f using Monte Carlo estimator with k samples [Haines and Akenine-Möller, 2019].

$$E\left[\frac{1}{k}\sum_{i=1}^{k} f(X_i)\right] = \int_{[0,1]^n} f(x)\ dx \tag{1.2}$$

Inside the expected value is an average of values of f using a set of independent uniform variables X_i on $[0,1]^n$.

This estimator is very simple and ineffective. Haines and Akenine-Möller [2019] have shown that it is better to sample rays nonuniformly using a distribution $p(x)$ which matches the function $f(x)$ as closely as possible. The following Equation 1.3 incorporates the nonuniform distribution, slightly changing from the previous Equation 1.2.

$$E\left[\frac{1}{k}\sum_{i=1}^{k} \frac{f(X_i)}{p(X_i)}\right] = \int_{[0,1]^n} f(x)\ dx \tag{1.3}$$

1.1.6 Photon Mapping

Photon mapping is a global illumination method for rendering indirect lighting and caustics [Haines and Akenine-Möller, 2019]. It is similar to bidirectional path tracing, although instead of connecting an eye path to a light path, it estimates the light arriving at the point P by collecting information from all the light paths. Since there may not be any light paths that end exactly at the point P, the method involves estimating the arriving light by examining neighbouring points and interpolating data [Hughes, 2014].

The incoming light is stored in a photon map, describing radiance at different points of the scene. The radiance is estimated with scattered indirect light of diffuse surfaces. Storing a sample of the arriving light allows for representing many outgoing light rays, so there is no need to trace them all individually [Hughes, 2014].

This sample is what is meant by *photon* in photon mapping. Following bounces of the photon are also recorded until it is absorbed or the recursion depth limit is exceeded. Hughes [2014] describes it as "a bit of power emitted by the light, typically representing many physical photons [per second]." It is defined with a location in space, a direction towards the light source or the last light bounce, and an incident power [Hughes, 2014].

Image Space Photon Mapping

McGuire and Luebke [2009] have presented a real time approach to photon mapping,

which operates with photons as volumes in image space. The algorithm works only with points lights and pinhole cameras, bringing a lot of limitations. On the other hand, it is significantly sped up compared to the original photon mapping approach [Hughes, 2014].

Screen Space Photon Mapping

Traditional photon mapping requires a lot of ray tracing to produce smooth light images and thus is not suitable for real-time rendering [Haines and Akenine-Möller, 2019]. However, there have been proposed real time solutions. Haines and Akenine-Möller [2019] introduced a technique called screen space photon mapping (SSPM), where photons are stored in screen space texels. This algorithm produces reflected and refracted caustics, although without global illumination of the whole scene.

Firstly, the photons are emitted and ray traced in the world space. Then, they are stored in a screen space texture representing the photon map. Secondly, a process called photon gathering has to be applied to remove noise in the caustics. Its result is a denoised image which can be used together with direct lighting to render a final image. This approach uses deferred rendering system. For detailed explanation of tracing the photons and their gathering, see the book Ray Tracing Gems by Haines and Akenine-Möller [2019].

Adaptive Anisotropic Photon Scattering

Adaptive Anisotropic Photon Scattering (AAPS) is a novel real time method proposed by Yang and Ouyang [2021] in the book Ray Tracing Gems II. To make the rendering interactive, it utilizes hardware ray tracing, similarly as SSPM. AAPS builds upon a variation of photon mapping, photon splatting and produces high-quality results with less blurry and noisy caustics than the previous approach, SSPM. Additionally, this method allows for rendering dispersion and soft caustics.

The algorithm maintains multiple buffers [Marrs et al., 2021].

- *Task buffers* store which photons are to be traced in the current frame.
- A *photon buffer* contains data of each photon, including its hit position, intensity and footprint.
- *Feedback buffers* consist of textures in the light space, managing feedback information over a couple of last frames.
- A *caustics buffer* contains a render target where photons will be splatted in screen space.

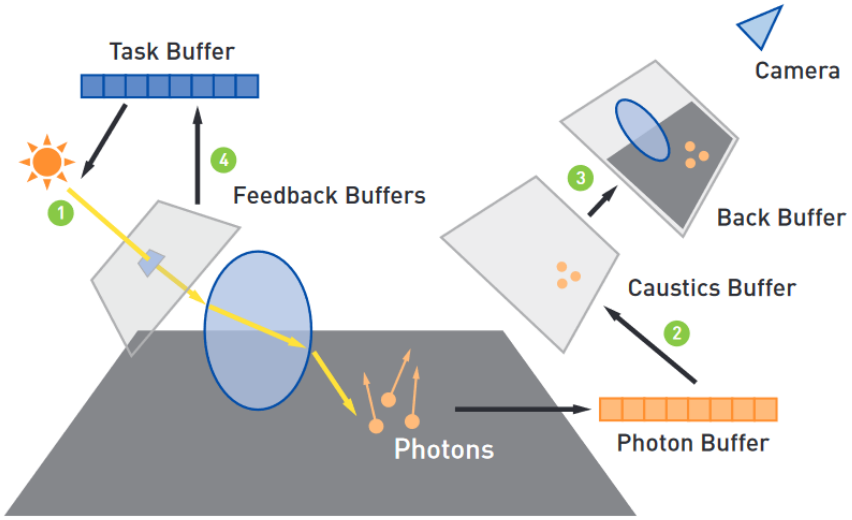

Figure 1.3: *Diagram of Adaptive Anisotropic Photon Scattering.* Rendering with AAPS is split into four steps [Marrs et al., 2021].

1. Emit photons from the task buffers. Trace them throughout the scene and for each hit on an opaque surface add details to the photon buffer and photon footprint to the feedback buffers.

2. Execute photon scattering (splatting). From photon buffer draw each photon as an elliptical footprint.

3. Apply the caustics buffer: carry out deferred lighting.

4. Create task buffers for the next frame by merging feedback buffers of the current and the previous frame.

From the book Ray Tracing Gems II, chapter 30 by Marrs et al. [2021].

Figure 1.3 shows details of the algorithm including the use of the buffers.

This novel method still operates in the screen-space so there are limitations, e.g. in caustics reflected by a mirror. It also neglects roughness values and the dispersion effects are not physically-accurate.

1.2 Caustics

Caustic is a curve of light caused by reflection and refraction of light shined on a transparent or a translucent object [Yang et al., 2021]. Metallic objects can also create this phenomenon. Hughes [2014] elaborates more and states that when the light photons interact with curved surfaces, they are focused into concentrated light curves.

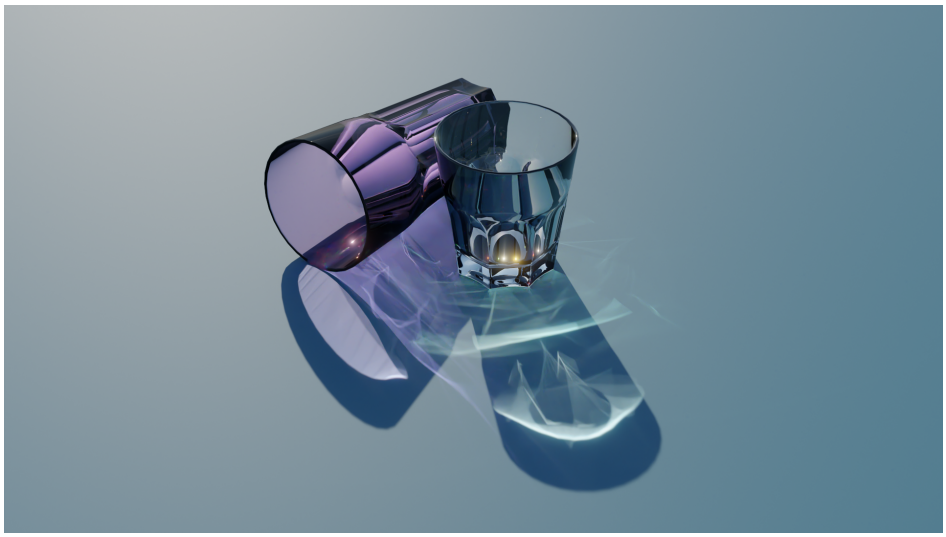

Figure 1.4: *Example of caustics beneath drinking glasses.* The reflected and refracted light rays concentrate into patches of light with intense bright edges sometimes forming cusp singularities. *Rendered in LuxCoreRender (path tracing).*

Hughes [2014] also adds that caustics are produced by point lights, small area lights and sunlight. They disappear under diffuse lighting.

Figure 1.4 shows an example of caustic patterns usually seen under drinking glasses.

Physically-accurate simulation of caustics is computationally expensive and simple rasterization rendering engines are unable to produce them. To simulate these effects we need to trace light bounces and thus use a path tracing or a raytracing engine.

Caustics can be rendered in a simplistic way using tricks. However, to bridge the Sim2Real gap we aim to use hardware ray tracing to achieve more accurate illumination in real time. We have mentioned one such method in Section 1.1, called AAPS.

Chapter 2

Existing Tools

In this chapter we name well-known software useful for our tasks and also previous work concerning similar problems. We shortly describe Blender, Unreal Engine and its experimental version made by Nvidia. Finally, we list a few plugins and previous solutions of synthetic data generation.

2.1 Blender

Blender is a free and open-source 3D program featuring tools for the whole 3D content-creation pipeline. It is licensed under GNU GPL and it is being developed by an online community and Blender Institute. It has an API for python scripting, making it easily extendable. Blender is cross-platform - runs on Linux, Windows and Macintosh.

A new tool called Geometry Nodes has been added to the software in recent updates. It features a node-based visual scripting for procedurally generating 3D objects and scenes. Using this tool, we can, for example, easily construct and modify geometry of meshes.

2.2 Unreal Engine

Unreal Engine is a game engine developed by Epic Games and it is popular among the game development community. The engine receives regular updates oftentimes introducing state-of-the-art implementations of new features and thus has a role of a technological leader in the game industry.

Unreal Engine's vast community consists not only of game developers. With many years of its existence, the engine has evolved into a large platform for digital content creation in general, which interests experts from various fields. Apart from making games it is also used for architectural visualizations, automobile design, product design, simulations and films.

Nvidia's NvRTX Branches

Nvidia tightly cooperates with Epic Games to improve rendering capabilities and introduce new features. Nvidia also implements its own technologies into the engine, such as DLSS, Frame Generation, Reflex and NRD.

Those which are still experimental are developed in their fork of Unreal Engine on GitHub. There is a specific branch for implementing caustics rendering which we aim to explore (NvRTXT caustics branch). [1] These branches are publicly accessible but require a registration.

2.3 Related Work

There are multiple existing tools regarding real-time rendering of synthetic data, developed for use with Unreal Engine. Its powerful rendering capabilities and ability to extend features through plugins shows why it is a popular choice among computer vision researchers.

SuperCaustics

This tool extends the NvRTX Caustics 4.26 branch of Unreal Engine and has been made by Mousavi and Estrada [2021]. Its purpose is to generate synthetic datasets of translucent objects with caustics. It is fully automated, generating random scenes in the editor and rendering them using hardware ray tracing.

We could not test the tool, since the particular version of the NvRTX Unreal Engine branch cannot be installed properly because it relies on downloading dependencies from a server with forbidden access. Nonetheless, it possesses a limitation, which we would like to overcome - the materials cannot be randomized during the scene generation process.

UnrealGT

UnrealGT is an Unreal Engine 4 plugin which provides a user with the ability to generate generic image datasets inside the editor. It was develop by Pollok et al. [2019] and provides some additional features, e.g. segmentation of the scene by configuring object classification. The user can thus render not only color images but also segmentation masks and even depth images. Other ground truth data like bounding boxes and pose data can be obtained too.

[1] The Unreal Engine version we have used is accesssible here.
`https://github.com/NvRTX/UnrealEngine/tree/NvRTX_Caustics-5.2`

It works only in Unreal Engine 4 and is not compatible with Unreal Engine 5 and later versions. Unreal Engine has received a lot of updates in version 5, including rewriting its code, in spite of which some classes have been refactored. That makes it impossible to compile this plugin inside Unreal Engine 5.

UnrealCV

UnrealCV, created by Qiu et al. [2017], is an Unreal Engine plugin which helps computer vision researchers create synthetic data, similarly to UnrealGT above (for example object masks; depth and normal images). Compared to the tools above, it runs on versions of Unreal Engine up to 5.2, so it is still being maintained.

It operates through a set of console commands inside the editor, allowing interactions with the virtual world. With them a user can change the transforms of a camera or render the current scene. Alternatively, one can use their client API and connect to the editor from an external program.

EasySynth

EasySynth is an Unreal Engine plugin working up to version 5.2., which helps with generating images for machine learning. It is a widget inside the Unreal Engine editor, which helps with export configurations for the rendering process. It allows to export RGB, depth, normal and optical flow images and also camera poses. There is a tool for object masking which allows to also generate semantic images.

SynBin

This work by Kravár et al. [2023] has explored possibilites of procedurally generating meshes inside Blender. They proposed a pipeline for creating cardboard boxes with Geometry Nodes system. These models served as synthetic data for training neural network to 6D pose estimation task, which proved to be superior to previous training attempts.

Chapter 3

Implementation

3.1 Methods

3.2 Parametric 3D Models

3.3 Parametric Materials

3.4 Rendering Synthetic Data

3.5 Automating the Process

Chapter 4

Results

4.1 Image Quality Evaluation

4.2 Synthetic Dataset

Conclusion

Bibliography

Cook, R. L., Porter, T., and Carpenter, L. (1984). Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145.

Glassner, A. S. (1989). *An introduction to ray tracing*. Morgan Kaufmann.

Haines, E. and Akenine-Möller, T., editors (2019). *Ray Tracing Gems*. Apress. `http://raytracinggems.com`.

Hughes, J. F. (2014). *Computer graphics: principles and practice*. Pearson Education.

Kajiya, J. T. (1986). The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150.

Kravár, P., Gajdoech, L., and Madaras, M. (2023). Novel synthetic data tool for data-driven cardboard box localization. In *International Conference on Artificial Neural Networks*, pages 565–569. Springer.

Lafortune, E. P. and Willems, Y. D. (1993). Bi-directional path tracing.

Marrs, A., Shirley, P., and Wald, I. (2021). *Ray tracing Gems II: next generation real-time rendering with DXR, Vulkan, and OptiX*. Springer Nature.

McGuire, M. and Luebke, D. (2009). Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 77–89.

Mousavi, M. and Estrada, R. (2021). Supercaustics: Real-time, open-source simulation of transparent objects for deep learning applications. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 649–655. IEEE.

Pollok, T., Junglas, L., Ruf, B., and Schumann, A. (2019). Unrealgt: Using unreal engine to generate ground truth datasets. In *Advances in Visual Computing : 14th International Symposium on Visual Computing, ISVC 2019, Lake Tahoe, NV, USA, October 7–9, 2019, Proceedings, Part I. Ed.: George Bebis*, page 670–682. Springer.

Qiu, W., Zhong, F., Zhang, Y., Qiao, S., Xiao, Z., Kim, T. S., Wang, Y., and Yuille, A. (2017). Unrealcv: Virtual worlds for computer vision. *ACM Multimedia Open Source Software Competition*.

Shirley, P., Ashikhmin, M., and Marschner, S. (2009). *Fundamentals of computer graphics*. AK Peters/CRC Press.

Veach, E. and Guibas, L. (1995). Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*, pages 145–167. Springer.

Yang, F., Wünsche, B. C., and MacDonald, B. (2021). Real-time caustics and dispersion on arbitrary surfaces in gpu-accelerated ray tracing. In *2021 36th International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 1–6.

Yang, X. and Ouyang, Y. (2021). *Real-Time Ray Traced Caustics*, pages 469–497.